

Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments

Rick Kazman

Department of Computer Science, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

1 The Need for Distribution

Stalin once remarked that “quantity has a quality all its own”. This philosophy is beginning to take hold in most computation-intensive fields, where fast, expensive supercomputers are beginning to be replaced by networks of smaller, cheaper computers. Although many researchers are designing and building virtual worlds based upon the functionality of a single hardware platform or a small number of tightly connected platforms, the need for distributing virtual worlds has been well demonstrated in domains such as flight simulators [9] and distributed tank combat simulations [2]. As soon as developers attempt to build virtual worlds of real-world fidelity, the need to distribute and scale up becomes apparent. Current flight simulators require dozens of CPU working in concert, and a flight simulator is potentially a single participant in a virtual world.

In addition, different hardware platforms are best suited for different tasks in simulating a virtual world (for reasons of speed, economy, availability of software and/or hardware, compatibility with I/O devices) and so any distributed virtual world must be prepared to support communication among a large and heterogeneous set of software and hardware devices. Finally, by developing a scalable environment for virtual worlds based upon heterogeneous platforms, researchers can utilize existing hardware, and so can begin to do research without a large capital outlay.

For these reasons, it is imperative to explore the architectural constraints placed on a virtual world by distribution and parallelism. In particular, we examine what it means to distribute functionality such as simulation, interaction detection and messaging in a virtual world, how to “scale up” such a world, and how to deal with communication delays.

2 Current Approaches to Distribution

Current approaches to architectures for virtual world rely upon either a client-server architecture, or upon a static allocation of processes to processors, which communicate among each other along predetermined paths [4] [5] [12] [13].

These architectures pose two potential problems for scale-up:

1. Point to point models fall apart when object communication is many-to-many. That is, in a simulation with 100 independent but closely interacting entities a point-to-point communication model would mean potentially sending 10,000 distinct messages at each time slice in order to handle interactions properly.

2. In broadcast models, much of the information being communicated between participants is potentially unnecessary in a large virtual space where any object in the virtual world may only need to know about other objects in its immediate vicinity, and so communication bandwidth is generally wasted.
3. In hierarchical structures, communication between processes is limited to the parent-child relation. In a complex virtual world, it is obvious that the root of the tree would quickly become a bottleneck, since that process is where all control eventually resides.

It is apparent, from examining the approaches taken in the architectures above that more flexible approaches to managing, integrating and communicating between processes is necessary. The WAVES (WATERloo Virtual Environment System) architecture attempts to address these concerns.

3 WAVES System Architecture

A virtual world, in this model, is comprised of a number of *Message Managers* which mediate communication between a number of *Hosts*. Hosts are processes which simulate a number of objoids¹ in the virtual world, and provide some services to those objoids—namely simulation, data logging, interaction detection and resolution, “switchboard” routing and rendering. The virtual world is composed of nothing but objoids. An architectural description of a typical WAVES cluster is given in figure 1.

In this picture, a message manager is represented as a double rectangle, hosts are represented as ovals and I/O devices are represented as single rectangles. Communication between these entities is either direct (i.e., connected by a serial line or LAN, represented as a straight line in figure 1) or through a communication channel such as TCP/IP sockets, represented by a “lightning-bolt” style line. Note that some hosts are attached to I/O devices but this is not a necessary feature of a host. Users interact with the system through devices attached to hosts. In some cases this device interaction has no effect on the world, such as when a user turns his head—in this case, the rendered attached to a host must update the scene being shown to the user, but the state of the world is unaffected. In other cases, the user might directly control an objoid in the world—say a virtual car which the user is driving—and any changes to this car’s state would have to be reflected wherever this car objoid existed.

In a distributed virtual world, each host only simulates a subset of the objoids in the virtual world, and so a collection of hosts is required to create an entire simulated world. Some of these objoids will be native to a particular host and potentially under the control of a user associated with that host, and some objoids will be *clones* of objoids which are native to other hosts, and which have been communicated to the user’s host in order to provide a realistic shared simulated world.

The message manager mediates communication between hosts and alleviates from individual hosts the responsibility of knowing who to communicate with, and when. This allows

¹The somewhat ugly term *objoid* is used, rather than the more familiar *object* because object carries too much baggage with it from object oriented programming, in the form of preconceived notions about inheritance, polymorphism and so on. An objoid is simply an encapsulation of state, a set of exported attributes, and a number of executable behaviors which can update the internal state autonomously.

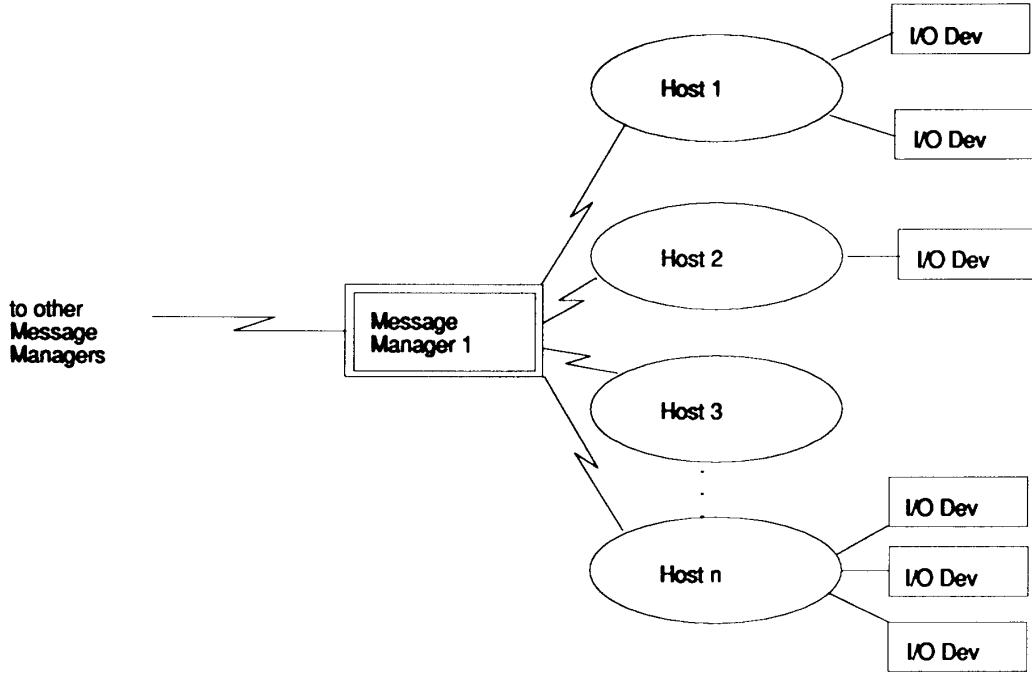


Figure 1: The WAVES Architecture

for flexible virtual worlds, where hosts can enter and leave dynamically, and allows for a smooth scaling of world complexity, because hosts do not logically depend on each other.

The message manager, in addition to setting up and removing virtual connections between hosts, can, upon request, filter the messages which a particular host receives. Such a function is crucial to minimizing levels of communication in a complex world, where most events will not be of interest to any given host, and so a simple message broadcasting strategy would not be appropriate. For example, in a large virtual world (such as SIMNET [3] or a large building rendered for an architectural walkthrough [6]), any given user will only view a small subset of the objoids in the virtual world. Because of the sparseness of information which any user needs to see at any given time (compared with all the information contained in a virtual world) intelligent filtering can drastically reduce the message volume in a system.

Hosts, when they initialize, and at any other time in their life, can send a set of filter specifications to their message manager. These filters are either semantic (based on particular attributes, i.e. only send me objoids that make a sound) or positional (only send me objoids which are located in this particular region of space).

The message manager can not only transmit messages from one host to others, but can, in special cases where data volumes are high and minimizing latency is crucial (for example, a direct video feed), delegate point-to-point communication between hosts. The message manager, for large virtual worlds will, in fact, be a family of distributed, interconnected managers, where the allocation of hosts to particular managers will be accomplished by a dynamic clustering algorithm, described elsewhere [8].

Communication between the message manager and hosts is principally in the form of requests for services, things like “establish a connection”, “remove a connection”, “send information which meets the following semantic criteria”, etc. Communications between objoid hosts are all based around parameterizable objoid specifications which hosts execute: hosts must exchange messages—through the medium of the message manager—such as “create a new objoid”, “destroy an objoid”, “acknowledge a message”, and “update an objoid’s attribute” in order to keep each other apprised of the current state of the world.

4 Models of Behavior

Objoids, in the WAVES architecture, are viewed as being autonomous entities which encapsulate their own state and possess an explicit model of behavior. Using this behavioral model, objoids can update their state in real time. In fact, behavioral models may be used to predict an objoid’s state, in order to accommodate system latencies. Having explicit encapsulated models of behavior offers several advantages for a distributed virtual world as has been shown in [7] and [2]:

- Objoids can be more easily transported from host to host, because their state and their computation are explicitly represented as attributes.
- Objoids—the original and its clones—can be simulated on a large number of distinct hosts and as long as their behavior does not significantly deviate from what their attributes and behavioral model predict, there is no communication necessary among these clones. This significantly cuts down on the potential message load in a distributed virtual world. Instead of having to report individual changes to an objoid, the only changes which need to be reported are those which deviate significantly from what would be predicted by the behavioral model.
- Explicit behavioral models can be used to mitigate the effects of communication latency by *predicting* an objoid’s behavior into the future, and anticipating behavioral changes far enough in advance that they appear to happen instantaneously (from the perspective of the user). For example, the use of behavioral models in interaction detection will be discussed below.

Current approaches to limiting communication have focussed on first and second order approximations of behavior, rather than on explicit behavioral models. For example, SIMNET [3] used dead-reckoning models of behavior to predict tank movement into the future and Kalman filtering techniques have been used to predict head-tracking data [10].

To give a musical analogy, first order prediction techniques, given that an E \flat was just heard, could only predict that an E \flat will continue to be heard. This is the equivalent of a dead-reckoning model. Using a second order technique, upon hearing the sequence C, D, E \flat , might predict that it was hearing a C minor scale, and so would predict F, G, A \flat , and so on. This is the analogic equivalent of a Kalman filter model. A behavioral model, on the other hand, would know the melody being played, and so could predict the entirety of Beethoven’s 3rd piano concerto. Using such a model, objoid behaviors can be simulated simultaneously

on numerous sites with little or no communication between them. This, of course, assumes that the behavior is fairly regular, i.e. that the pianist doesn't spontaneously begin to speed up, or improvise or play incorrect notes.

Even in problematic situations such as these, however, some repair strategies can be invoked. For example, if the pianist suddenly begins playing Rhapsody in Blue, the new behavior will be announced, and an appropriate behavioral model—the Rhapsody in Blue model—can be simulated. If the pianist is playing Chopin, where *rubato*—frequent changes in tempo—is frequently employed, clones of the pianist objoid can make successive approximations to the new tempo over the course of the next few simulation frames, thus presenting a consistent environment to the user.

The success of behavioral models is thus directly linked to the predictability of the behavior being simulated. Fortunately, most simulated behavior appears to be relatively long periods of predictable behavior, punctuated by occasional bursts of unpredictable behavior. In worlds such as this, behavioral models can significantly reduce the amount of communications needed. Since communications is typically the bottleneck in any distributed application, this is a significant advantage.

5 Separation of Concerns

Although objoids in the WAVES architecture possess a model of behavior, they do not control their rendering in the world. In this way, we maintain the crucial distinction between functionality (content) and presentation (style) found commonly in UIMS architectures [1] and in some virtual world architectures [5]. Furthermore, objoid specifications are not constrained to a particular implementation. They simply specify a parameterizable real-time behavior, and potentially export a number of attributes by which this behavior can be controlled.

Hosts can (but need not necessarily) control some input or output device. In addition, hosts can serve other purposes—for example, hosts can listen to the state of the virtual world for the purposes of interaction detection. Because interaction detection is such a complex task in any complex world, it too must be distributed. Interaction detection hosts are specialized hosts in the WAVES architecture. They control no I/O device, but simply receive objoid specifications, and produce notifications of interactions between objoids in the simulated world. Furthermore, for certain types of interactions (as outlined in [7]), interaction detection hosts can resolve the conflicting behavior. This is done by modifying the behavior of the conflicting objoids (i.e., modifying the objoids' exported attributes). Such a task is called *Interaction Resolution*. Interaction resolution is often better handled outside the objoid for reasons of generality. Consider the following problem: if objoids A and B interact, where should the resolution code go? In A, in B or in both? Furthermore, if there are 50 different objoid types in the virtual world, does each objoid type have to have code specific to resolving interactions with each other type? Clearly, a case can be made here for centralization of functionality.

Hosts and the message manager can, and typically will, maintain a cache of objoids which they are currently interested in, and can update this cache at any time. Without such a cache, the WAVES architecture devolves to a distributed server architecture with broadcasting as the only mechanism of keeping the world consistent.

To summarize, the amount of inter-objoid communication in the WAVES is greatly reduced (as compared with the classical client-server or point-to-point paradigms). When an objoid's behavior changes, hosts which are interested in that objoid are notified (and only those hosts are notified, because of the filtering mechanism built in to the message manager) and hosts can update their objoid cache to reflect these changes, and continue simulating their objoids. A virtual world is thus a set of cooperating processes which can communicate by virtue of subscribing to a communication service. Its structure is completely determined by the dynamics of the simulated world, and so it can be easily scaled by adding more hosts, objoids, and message managers.

6 Implementation

A prototype of the WAVES architecture, called HIDRA [7] has been implemented as both a proof of concept and as a way of investigating the interplay between autonomous behavior, interaction detection and resolution, and the inherent communication delays which are a necessary evil of such a distributed system. This prototype has been implemented in C, running on the Sun Sparc 2 architecture, and contains the components listed in figure 1: a message manager and objoid hosts, which include specialized hosts such as a world view maintainer (not discussed here), and interaction detection/resolution servers. This implementation did not include many of the features for significantly reducing message volumes (intelligent message filtering) and for flexibility (load balancing) which are necessary in large, scalable, complex virtual worlds.

A second version of the WAVES architecture is now being designed and implemented which will concentrate on these additional features as well as providing support for a heterogeneous set of software and hardware devices. This architecture is initially being targeted for IBM 386/486 PCs (building upon the success of REND386 software [11]) IBM RS/6000 workstations, Sun Sparc 2 workstations and Silicon Graphics Indigos and Onyxes. We expect to make this release publicly available by late 1993.

References

- [1] L. Bass, B. Clapper, E. Hardy, R. Kazman, and R. Seacord. Serpent: A user interface management system. In *The Proceedings of the Winter 1990 USENIX Conference*, pages 245–257, 1990.
- [2] B. Blau, C. E. Hughes, J. M. Moshell, and C. Lisle. Networked virtual environments. In *ACM SIGGRAPH 92*, pages 157–160, 1992.
- [3] R. A. Brooks, B. G. Buchanan, D. B. Lenat, D. M. McKeown, and J. D. Fletcher. Panel review of the semi-automated forces. Technical Report IDA Document D-661, Institute for Defense Analysis, Alexandria, Virginia, 1989.
- [4] G. P. Coco. VEOS 2.0 tool builders manual. Unpublished paper, 1992.
- [5] C. Codella, R. Jalili, L. Koved, B. Lewis, D. Ling, J. Lipscomb, D. Rabonhorst, C. Wang, A. Norton, P. Sweeney, and G. Turk. Interactive simulation in a multi-person virtual world. In *Proc. of CHI '92*, pages 329–334, 1992.

- [6] T. A. Funkhouser, C. H. Sequin, and S. J. Teller. Management of large amounts of data in interactive building walkthroughs. In *ACM SIGGRAPH 92*, pages 11–20, 1992.
- [7] R. Kazman. Hidra: An architecture for highly dynamic physically based multi-agent simulations. *International Journal in Computer Simulation*, 1993. (To appear).
- [8] R. Kazman. Load balancing and latency management in a distributed virtual world. Paper presented to the 3rd International Conference on Cyberspace. Austin TX, May 14-15 1993.
- [9] K. Lee, M. Rissman, R. D’Ippolito, C. Plinta, and R. Van Scy. An OOD paradigm for flight simulators, 2nd edition. Technical Report CMU/SEI-88-TR-30, Carnegie Mellon University, 1988.
- [10] J. Liang, C. Shaw, and M. Green. On temporal-spatial realism in the virtual reality environment. In *UIST ’91: Proc. of the ACM Symposium on User Interface Software and Technology*, pages 19–26, 1991.
- [11] B. Roehl and D. Stampe. *Virtual Reality Creations*. Waite Group, 1993. (To appear).
- [12] C. Shaw, J. Liang, M. Green, and Y. Sun. The decoupled simulation model for virtual reality systems. In *Proc. of CHI ’92*, pages 321–328, 1992.
- [13] A. J. West, T. J. J. Howard, R. J. Hubbald, A. D. Murta, D. N. Snowdon, and D. A. Butler. Aviary—a generic virtual reality interface for real applications. Unpublished paper, 1992.